Where does it go from here? The place of software in digital repositories

Neil P. Chue Hong
Software Sustainability Institute, University of Edinburgh
N.ChueHong@software.ac.uk

Abstract

The open repositories community has made great strides in recent years in addressing interoperability, policy and providing the arguments for open access and sharing. One aspect of open research which has come to prominence is the importance of software as a fundamental part of reproducible research, which in turn raises issues around the preservation of software.

In this short paper, I will describe some of the work that the Software Sustainability Institute (SSI) has been doing to address the structural and policy issues which currently present a barrier to the deposit and use of software in open repositories.

1. Introduction

In recent years, scientific research has been augmented by the use of computer software for data capture, simulation, data analysis, and more. With the advent of what has been termed "e-Science", it has increasingly focused on the use of software carried out by communities of researchers that span disciplines, laboratories, organisations, and national boundaries.

This permeation of the use of software into the mainstream of research across all disciplines has meant that it is increasingly difficult to reproduce and reuse the work of other researchers. The reproducible research principle requires the full computational environment to be published as well as the paper where the results are reported.

This raises the question of whether the current preservation policies and digital repositories are capable of handling software as a digital object that requires preservation.

2. Software Preservation

A key challenge in digital preservation is being able to articulate, and ideally prove, the need for preservation. A clear framework of purposes and benefits facilitates making the case for preservation. This is even more relevant for the specific case of software preservation. In the JISC funded *Clarifying the Purpose and Benefits of Preserving Software* project[1] the SSI and Curtis+Cartwright developed a benefits framework [1] which identified four key benefits to preserving software:

- Encourage software reuse;
- Achieve legal compliance and accountability;
- Create heritage value;
- Enable continued access to data and services.

It also identified seven different approaches to software preservation and sustainability:

- *Technical preservation (techno-centric)* - Preserve original hardware and software in same state;
- *Emulation (data-centric)* - Emulate original hardware / operating environment, keeping software in same state;
- *Migration (functionality-centric)* - Update software as required to maintain same functionality, porting/transferring before platform obsolescence;

[1] Project website: http://softwarepreservation.jiscinvolve.org/

- *Cultivation (process-centric)* - Keep software 'alive' by moving to a more open development model, bringing on board additional contributors and spreading knowledge of process;
- *Hibernation (knowledge-centric)* - Preserve the knowledge of how to resuscitate/recreate the exact functionality of the software at a later date;
- *Deprecation* - Formally retire the software without leaving the option of resuscitation/recreation;
- *Procrastination* - Do nothing.

Two of the benefits – achieving legal compliance and accountability and enabling continued access to data and services – pose relevant questions for the open repository community and are driven very much by the *reproducible research* principle.

Reproducible research [3] refers to the idea that the ultimate product of research is the traditional paper output along with the full computational environment used to produce the results in the paper such as the code, data and methods necessary for reproduction of the results and building upon the research. Whilst great strides have been made to ensure that we can preserve the data sets along with the paper, less has been done in terms of understanding how to preserve the software and its related environment (with some notable exceptions [4] [7] [9]).

We perceive that there are two principal challenges when considering the preservation of software in a fixed state (i.e. technical preservation, emulation and hibernation) for the medium-to-long term for the purpose of reproducible research: accurately capturing dependencies; and understanding how to reference and credit the deposited software upon reuse.

3. Significant properties and software metadata

Previous work undertaken by STFC [4] [6] has looked at identifying the significant properties of software, that is the essential attributes of the software which affect its appearance, behaviour, quality and usability. Significant properties must be preserved over time for the software object to remain accessible and meaningful. Moreover, because the boundaries of a piece of software are less distinct than a dataset, the dependencies – both explicit and tacit, functional and non-functional become more important. There is also great scope for ambiguity in the internal contents of a "software object". Examples of this include: alternative routines and libraries; downloadable content (DLC) which is accessed post "release".

Work at the SSI has involved both assessing how easy a framework such as that described in [4] could be implemented and utilised given the above issues, as well as participating in efforts such as SWOP[2] to develop a vocabulary that will help describe software used by the curation and data preservation community with a secondary outcome of better understanding how the curation community are using software descriptions within their preservation work.

4. Access, citation and credit for software

Much has been done to improve the accessibility to and ability to reference datasets, in particular through the DataCite[3] and ORCID[4] initiatives.

Nevertheless, there are a number of issues [2] that make software a particularly specialised form of dataset when it comes to citations and harder to reuse the work done by others pertaining to datasets. Many, such as versioning and authorship, are ameliorated in the case of specific deposit in relation to a publication, however the issue of *granularity* remains.

In general, when a dataset is deposited in a repository, that specifies its granularity – the dataset can be considered as a unique object. It may consist of a collection of pieces of data that have distinct characteristics (e.g. an album is well-defined as a collection of songs) but importantly there is a point in time at which the distinction is made. Software has become harder to define, in particular as software has made the leap from a single machine to a distributed system. What do you consider a part of the software, and what might you assign an identifier

[2] Software Ontology project: http://softwareontology.wordpress.com/
[3] DataCite: http://datacite.org/
[4] ORCID: http://about.orcid.org/

to? What is an "application": source code, binaries, workflows, manuals, website? And how much should we consider in terms of dependencies such as operating systems and hardware characteristics?

5. Digital Repositories and Software

It is clear that there are two separate concerns when looking at the preservation of software to enable reproducible research. Firstly, the software must be deposited somewhere where the depositor can both have reasonable assurances of persistent storage, with appropriate metadata to allow for retrieval and replay. Secondly, there must be mechanisms in place to reference this instance of the software and connect it to the other digital ephemera to which it is associated such as the papers, datasets and workflows.

For storage, whilst there are many publicly available providers of code repositories (e.g. SourceForge[5], GitHub[6]) these may not provide the necessary assurances for long-term storage. Likewise these repositories are often structured and organised to support the constant evolution and development of the code, which is not appropriate for "stored" code versions. There are some generally available filestores which would be appropriate for software storage (e.g. FigShare[7]) and some which might be repurposed to allow software storage (e.g. Dryad[8]). There are many mature, open, technical approaches to persistent, guaranteed storage of data (e.g. LOCKSS[9] [4], dSPACE[10], Fedora[11]) however it is not appropriate for each researcher developing software to operate such a repository themselves. Institutional repositories would also appear to be the correct place for researchers to deposit software as they are already starting to put in place policies to collect research outputs from projects, yet an analysis of the current policies of the major UK university repositories shows that they either ignore or exclude software deposit (UCL and Newcastle University are two which we are aware are running trials for deposit of software). This clearly needs to change, but a case must be made – as it was for data – of the benefits and also an analysis of the ongoing costs. Additionally, guidelines must be in place to ensure that appropriate metadata is recorded to allow reuse in the future, and that this is checked on some regular basis.

Ideally, the ability to reuse the existing mechanisms for referencing publications and datasets would also allow current bibliometric and impact measurement tools to be applied to software. An analysis of the guidelines and usage of Digital Object Identifiers (DOIs) for datasets and comparison with software only uncovers one barrier: how does the concept of a *data publisher*, also the body responsible for minting and administering the identifier, translate to software? It is likely that each software publisher (normally the research project, consortium or individual) is not in a position to generate DOIs on an ad-hoc basis. However, in the specific case of a software deposit to support reproducible research this barrier is easier to overcome. A single identifier is required to associate with the instance of the software deposited at the time of deposit of the related paper and datasets. Authorship can be fixed at the point of deposit. And the data publisher becomes the repository in which the author chooses to deposit the software, for instance Dryad already issues DOIs for deposited data packages.

A complementary approach to this, which uses existing mechanisms to allow for citable software is the idea of a *software paper*. In this scenario, the metadata and methodology required to reuse a piece of software is published as a "traditional" paper which is citable in the normal fashion. This software paper references the deposited software which can reside in a data repository such as FigShare or Dryad, an institutional repository, or even as a tagged version of the source code in a code repository (assuming persistence is addressed). This approach is currently being trialled by the SSI in conjunction with Ubiquity Press.

[5] SourceForge: http://www.sourceforge.net/
[6] GitHub: http://www.github.com/
[7] FigShare: http://figshare.com/
[8] Dryad: http://datadryad.org/
[9] LOCKSS: http://www.lockss.org/
[10] DSpace: http://www.dspace.org/
[11] Fedora: http://fedora-commons.org/

6. Conclusions

The increasing backing of the reproducible research ideals necessitates the ability to describe, store, retrieve and reuse software associated with publications. Work done to address similar requirements for datasets can mostly be reused for software in this special case as many of the issues pertaining to the rapid development, multiple dependencies, and assignment of identifiers are simplified. Nevertheless, even though many of the technical challenges have been solved, the policy – particularly at institutional repositories – needs to be revised to acknowledge the requirement to provide facilities for depositing software. This area is one which the Software Sustainability Institute is looking to address in collaboration with others.

7. Acknowledgements

The Software Sustainability Institute comprises staff of the universities of Edinburgh, Manchester, Oxford and Southampton, and is funded by the EPSRC under grant EP/H043160/1. The work on clarifying the purpose and benefits of preserving software was carried out in collaboration with Curtis and Cartwright Ltd and funded by the JISC. The work on SWOP is led by the University of Manchester and the EBI and is funded by the JISC.

8. References

1. N. Chue Hong, S. Crouch, S. Hettrick, T. Parkinson, and M. Shreeve, “Software Preservation Benefits Framework,” Software Sustainability Institute Technical Report, 2010. Retrieved on 5th March 2012 from: http://www.software.ac.uk/attach/SoftwarePreservationBenefitsFramework.pdf
2. N. Chue Hong, “Software Impact – the differences from datasets”, Beyond Impact blog, May 2011. Retrieved from: http://beyond-impact.org/?p=175
3. Sergey Fomel and Jon Claerbout, "Guest Editors' Introduction: Reproducible Research," Computing in Science and Engineering, vol. 11, no. 1, pp. 5–7, Jan./Feb. 2009, doi:10.1109/MCSE.2009.14
4. Petros Maniatis, Mema Roussopoulos, T. J. Giuli, David S. H. Rosenthal, and Mary Baker. 2005. The LOCKSS peer-to-peer digital preservation system. ACM Trans. Comput. Syst. 23, 1 (February 2005), 2-50. DOI=10.1145/1047915.1047917 http://doi.acm.org/10.1145/1047915.1047917
5. Matthews, B.M., McIlwrath, B., Giaretta, D., & Conway, E. (2008). The Significant Properties of Software: A Study. In JISC report, 2008. Retrieved August 3, 2009, retrieved from http://sigsoft.dcc.rl.ac.uk/twiki/pub/Main/SigSoftTalks/SignificantPropertiesofSoftware.doc
6. Brian Matthews, Arif Shaon, Juan Bicarregui, and Catherine Jones. 2010. A Framework for Software Preservation | Matthews | International Journal of Digital Curation. International Journal of Digital Curation 5, no. 1: 91-105. http://ijdc.net/index.php/ijdc/article/view/148, doi: 10.2218/ijdc.v5i1.145.
7. Piotr Nowakowski, Eryk Ciepiela, Daniel Harężlak, Joanna Kocot, Marek Kasztelnik, Tomasz Bartyński, Jan Meizner, Grzegorz Dyk, Maciej Malawski, The Collage Authoring Environment, Procedia Computer Science, Volume 4, 2011, Pages 608-617, ISSN 1877-0509, 10.1016/j.procs.2011.04.064.
8. Ricardo H Ramirez-Gonzalez, Raoul Bonnal, Mario Caccamo and Daniel MacLean, bio-samtools: Ruby bindings for SAMtools, a library for accessing BAM files containing high-throughput sequence alignments, Open Research Computation 2012, 1:1, doi:10.1186/2042-5767-1-1
9. Yale Law School Roundtable on Data and Code Sharing, "Reproducible Research," Computing in Science and Engineering, vol. 12, no. 5, pp. 8-13, Sep./Oct. 2010, doi:10.1109/MCSE.2010.113